**Indonesian Journal of Electronics and Instrumentation Systems (IJEIS)**
Vol.9, No.1, April 2019, pp. 65~74
ISSN (print): 2088-3714, ISSN (online): 2460-7681
DOI: 10.22146/ijeis.43906


# Implementasi Rangkaian CRC (*Cyclic Redundancy Check)* Generator pada FPGA (*Field Programmable Gate Array)*

**Nia Gella Augoestien*[1], Ryan Aditya[2]**
[1]Departemen Ilmu Komputer dan Elektronika, FMIPA UGM, Yogyakarta, Indonesia
[2] KSB Indonesia
e-mail: ***[1]nia.gella.a@ugm.ac.id**, [2]ryan.aditya@ksb.co.id

***Abstrak***

*Integritas data dari sebuah proses transmisi data kecepatan tinggi merupakan suatu syarat utama yang tidak dapat diabaikan. Transmisi data kecepatan tinggi rentan terhadap terjadinya kesalahan. CRC (Cyclic Redundancy Check) merupakan mekanisme yang sering digunakan sebagai pendeteksi kesalahan pada proses transmisi data dan penyimpanan data. Ketika CRC diimplementasikan menggunakan perangkat lunak atau prosessor tertanam, CRC membutuhkan waktu komputasi (siklus detak) yang sangat lama. Namun jika diimplementasikan menggunakan perangkat keras yang didedikasikan khusus, waktu komputasi yang dibutuhkan akan berkurang sehingga dapat memenuhi kebutuhan untuk sistem komunikasi kecepatan tinggi.*

*Paper ini mengusulkan rancangan dan implementasi rangkaian CRC 32 pada FPGA yang mampu meminimalkan waktu komputasi yang dibutuhkan. Cara yang ditempuh adalah dengan mengurangi latensi perhitungan dengan memisahkan koefisien digit-digit tertentu dan menghitung langsung hasil modulo dengan polinomial kunci yang digunakan.*

*CRC Generator pada penelitian ini diimplementasikan pada FPGA Xilinx Spartan®-6 Seri (XC6LX16-CS324). Hasil pemodelan telah berhasil melakukan komputasi dalam 1 siklus detak. Efisiensi perangkat keras yang dicapai adalah 0,38 Gbps/Slice, sedangkan lewatan yang diperoleh adalah 3,758 Gbps. Dengan pemakaian sumber daya hanya 10 slice (1%) dari yang tersedia pada FPGA.*

***Kata kunci****—komputasi kecepatan tinggi , pemrosessan detak tunggal, latensi rendah*

***Abstract***

*Data integrity in high speed data transmission process is a major requerment that can not be ignored. High speed data transmission is prone to data errors. CRC (Cyclic Redundancy Check) is a mechanism that is often used as a detector errors in data transmission and storage process. When CRC is implemented using embedded software or processor, CRC requires many clock cycles. If CRC Generator implemented in special dedicated hardware, computational time reduced so that it can be met the high speed system communication requirement.*

*This paper propose the design and implementation of CRC generator on FPGA that capable to minimaze computational time. The method is to reduce calculation latency by separating the coefficients of certain digits and calculating directly the result of polinomial key modulo.*

*CRC Generator in this paper was implemented on Xilinx Spartan®-6 Series (XC6LX16-CS324). The modeling results have succeeded to finish computation on 1 clock cycle. Hardware eficiency is achieved 0.38 Gbps/Slice, while the throughput is 3,758 Gbps.*

***Keywords****—High speed computation, single clock processing, Low latency*

*Received March 1st,2019; Revised April 19th, 2019; Accepted April 23th, 2019*

## 1. PENDAHULUAN

Dalam komunikasi data, kecepatan tinggi dan jarak yang jauh merupakan hal yang sangat dibutuhkan belakangan ini. Mempertahankan integritas dari data yang ditransmisikan merupakan syarat utama yang tidak dapat diabaikan. Masih sering terjadi kesalahan pada proses transmisi data dengan kecepatan tinggi dan jarak jauh [1]. Dengan demikian diiperlukan suatu metode untuk mendeteksi kemungkinan kesalahan dan memperbaiki kesalahan agar menghindari kesalahan dalam pengiriman data tersebut. Teknologi CRC (*Cyclic Redundant Check*) umumnya digunakan pada layer fisik protokol komunikasi[2]. Banyak protokol komunikasi dan penyimpanan yang menggunakan CRC sebagai metode pendeteksi kesalahan, yaitu; PCI Express (PCIe), Ethernet (IEEE 802.3) , WiFi (IEE 802.11), *Universal Serial Bus* (USB) dan *Double data rate fourth generation synchronous dynamic randomaccess memory* (DDR4 SDRAM) [3].

Menurut [4] CRC merupakan teknik untuk mendeteksi kesalahan yang sering digunakan karena kehandalannya. Selain itu komputasi CRC membutuhkan sumber daya yang terbatas dan mudah untuk diimplementasikan. Pada transmisi data, kode CRC dihasilkan untuk dimuat pada *frame check sequence* (FCS) dan dikirim bersama-sama dengan paket data. Piranti receiver menerima paket data dan menghitung CRC dari data yang diterima dan membandingkannya dengan kode CRC pada FCS [5].

Dalam implementasinya CRC seringkali dikombinasikan dengan metode deteksi kesalahan lain untuk menghasilkan transmisi data yang efisien seperti pada penelitian [6]. Walaupun kombinasi dengan metode lain mampu meningkatkan kinerja algoritma, namun terdapat beban komputasi baru berupa waktu ataupun resource tambahan yang perlu dipertimbangkan.

Dukungan komputasi yang cepat untuk menghasilkan kode CRC perlu menyesuaikan permintaan transmisi data kecepatan tinggi [4]. Ketika CRC diimplementasikan menggunakan perangkat lunak atau prosessor tertanam, CRC membutuhkan waktu komputasi (siklus detak) yang sangat lama. Namun jika diimplementasikan menggunakan perangkat keras yang didedikasikan khusus, komputasi yang dibutuhkan akan berkurang sehingga dapat memenuhi kebutuhan untuk sistem komunikasi kecepatan tinggi. FPGA *(Field Programmable Gate Array)* merupakan pilihan platform yang dapat digunakan untuk merancang dan mengimplementasikan rangkaian generator CRC. Fleksibelitas yang ditawarkan oleh FPGA karena dapat di konfigurasi ulang menjadikan FPGA sering digunakan untuk proses perancangan dengan tujuan meminimalkan sumber daya atau meningkatkan kecepatan komputasi. Komputasi yang cepat dapat dilakukan dengan meminimalkan latensi.

Metode umum yang sering digunakan untuk implementasi generator CRC pada FPGA adalah metode LFSR (*Linear Feedback Shift Register*) seperti yang dilakukan oleh [7] mengimplementasikan Algoritma CRC-16 pada FPGA untuk *Control Area Network* (CAN).Menurut [8] metode LFSR menggunakan arsitektur bit serial bit sederhana yang digunakan untuk mengimplementasikan algoritma CRC. Hal inilah yang mengakibatkan latensi akan bertambah banyak dengan meningkatnya derajat polinomial yang digunakan sebagai kunci. Dengan mengimplementasikan metode ini akan didapatkan pemanfaatan resource pada FPGA yang sangat minimal. Namun demikian untuk menghasilkan kode CRC menggunakan metode ini dibutuhkan latensi yang sangat banyak dan bertambah sesuai dengan derajat polinomial yang digunakan untuk menghasilkan kode CRC. Dengan bertambahnya latensi akan berpengaruh terhadap kecepatan dalam menghasilkan Kode CRC.Sumber daya FPGA yang digunakan pada penelitian [7] memang sangat sedikit, namun kecepatan maksimal yang dapat dicapai hanya 250kbps.

Pipeline dan paralelisme beberapa cara yang dapat ditempuh untuk meminimalkan latensi yang umum digunakan [1]. Dengan mengimplementasikan metode pipeline akan didapatkan nilai latensi yang tetap besar, namun metode ini dapat menghasilkan kecepatan

komputasi kode CRC yang lebih cepat karena memiki frekuensi maksimum yang tinggi. Sedangkan jika diimplementasikan menggunakan metode paralel, kecepatan komputasi kode CRC akan jauh lebih cepat, namun dengan penerapan metode ini akan berpengaruh pada pemakaian sumberdaya yang sangat besar dibandingkan 2 metode lain. Hasil implementasi dari metode ini juga membutuhkan memori ekstra untuk menyimpan nilai CRC dari komputasi sebelumnya dan data [9]. Selain itu, implementasi pada metode ini biasanya akan meningkatkan jalur kritis yang berpengaruh kepada frekuensi kerja maksimum perangkat keras.

Pada penelitian ini dirancang dan diimplementasi rangkaian generator CRC pada FPGA yang membutuhkan latensi minimal dalam 1 siklus yang ditempuh dengan memisahkan operasi setiap digit berdasarkan pada koefisien kunci polinomial yang digunakan. Selain itu pada implementasi pembagian dengan kunci polinomial operasi modulo akan dihitung langsung pada siklus yang sama.

## 2. METODE PENELITIAN

### *2.1. Perancangan sistem*

Kode generator CRC-32 dapat dihasilkan melalui 3 tahapan, yaitu; preprocessing, pembagian dengan kunci polinomial dan pastprocessing. Tahap preprocessing sendiri terdiri dari 3 operasi, yaitu;

- Membalik setiap input dari masukan
  Contoh: Jika input adalah karakter A dalam kode ASCII yang jika diterjemahkan dalam bentuk binner adalah “0100 0001” maka hasil dari operasi ini adalah “1000 00010”
- Menambahkan padding
  Penambahan padding dilakukan untuk menjamin input yang akan dihasilkan kode CRC memiliki panjang bit lebih dari 32bit sehingga dapat dihitung pembagian dengan kunci polinomialnya.
- Operasi XOR dengan oxFFFFFFF

Jika dilihat dari ketiga tahapan ini semua operasi dapat dilakukan secara kombinasional dengan urutan yang benar.

Tahapan pembagian dengan kunci polinomial merupakan tahapan inti dari proses komputasi untuk menghasilkan kode CRC. Proses pembagian sendiri pada tahapan ini sebenarnya dapat dilakukan dengan menggeser kiri inputan yang ada sebanyak 1 bit dan melakukan modulo terhadap kunci polinomial yang digunakan. Proses geser kiri identik dengan melakukan pekalian 2 terhadap masukan. Hal ini dilakukan berulang sampai didapatkan sisa hasil bagi sepanjang 32 bit.

Kode CRC yang akan dihitung dapat dihasilkan dalam satu siklus detak perlu dipasktikan bahwa proses yang ada harus dilakukan dalam 1 siklus. Dengan mengadopsi metode [10] pada komputasi transformasi mix column di algoritma AES diharapkan operasi pembagian ini dapat diselesaikan dalam 1 siklus. [10] memisahkan operasi modulo yang dilakukan berdasarkan koefisien digit-digit yang terdapat pada bilangan yang dijadikan modulo dan menghitung langsung hasil perkalian dengan 2.

Sebagai contoh,jika diketahui sebuah masukan **z** dengan panjang 8 bit dimana masukan tersebut didefenisikan sebagai Gambar 1, maka hasil operasi perkalian 2 terhadap **z** sekaligus modulo terhadap polinomial m(x) pada persamaan (1) adalah z’(x). Hasil **z’**(x)yang jika disajikan dengan memisahkan setiap digit koefisien pada Gambar 2.

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (1)$$

Masukan : **z**= { $z_7$ $z_6$ $z_5$ $z_4$ $z_3$ $z_2$ $z_1$ $z_0$ }

Gambar 1 Reprensentasi masukan **z**

| $z'_7$ | $z'_6$ | $z'_5$ | $z'_4$ |
|---|---|---|---|
| $z_6$ | $z_5$ | $z_4$ | $z_3 \oplus z_7$ |

| $z'_3$ | $z'_2$ | $z'_1$ | $z'_0$ |
|---|---|---|---|
| $z_2 \oplus z_7$ | $z_1$ | $z_0 \oplus z_7$ | $z_7$ |

Gambar 2 Hasil perhitungan perkalian 2 masukan dan modulo dengan polinomial m(x)

| $Z'_{31}$ | $Z'_{30}$ | $Z'_{29}$ | $Z'_{28}$ |
|---|---|---|---|
| $Z_{30}$ | $Z_{29}$ | $Z_{28}$ | $Z_{27}$ |

| $Z'_{31}$ | $Z'_{30}$ | $Z'_{29}$ | $Z'_{28}$ |
|---|---|---|---|
| $Z_{30}$ | $Z_{29}$ | $Z_{28}$ | $Z_{27}$ |

| $Z'_{27}$ | $Z'_{26}$ | $Z'_{25}$ | $Z'_{24}$ |
|---|---|---|---|
| $Z_{26}$ | $Z_{25} \oplus Z_{31}$ | $Z_{24}$ | $Z_{23}$ |

| $Z'_{23}$ | $Z'_{22}$ | $Z'_{21}$ | $Z'_{20}$ |
|---|---|---|---|
| $Z_{22} \oplus Z_{31}$ | $Z_{21} \oplus Z_{31}$ | $Z_{20}$ | $Z_{19}$ |

| $Z'_{19}$ | $Z'_{18}$ | $Z'_{17}$ | $Z'_{16}$ |
|---|---|---|---|
| $Z_{18}$ | $Z_{17}$ | $Z_{16}$ | $Z_{15} \oplus Z_{31}$ |

| $Z'_{15}$ | $Z'_{14}$ | $Z'_{13}$ | $Z'_{12}$ |
|---|---|---|---|
| $Z_{14}$ | $Z_{13}$ | $Z_{12}$ | $Z_{11} \oplus Z_{31}$ |

| $Z'_{11}$ | $Z'_{10}$ | $Z'_9$ | $Z'_8$ |
|---|---|---|---|
| $Z_{10} \oplus Z_{31}$ | $Z_9 \oplus Z_{31}$ | $Z_8$ | $Z_7 \oplus Z_{31}$ |

| $Z'_7$ | $Z'_6$ | $Z'_5$ | $Z'_4$ |
|---|---|---|---|
| $Z_6 \oplus Z_{31}$ | $Z_5$ | $Z_4 \oplus Z_{31}$ | $Z_3 \oplus Z_{31}$ |

| $Z'_3$ | $Z'_2$ | $Z'_1$ | $Z'_0$ |
|---|---|---|---|
| $Z_2$ | $Z_1 \oplus Z_{31}$ | $Z_0 \oplus Z_{31}$ | $Z_{31}$ |

Gambar 3 Bentuk perkalian dan modulasi dengan kunci polinomial G(x)

Jika dilihat dari hasil operasi dapat diketahui bahwa nilai z'1, z'3 dan z'4 adalah hasil operasi XOR bit sebelah kanannya dengan bit MSB. Sedangkan bit z'0 akan bernilai sama dengan MSB. Pola ini disesuaikan dengan digit-digit polinomial yang memiliki koefisien 1.

Dengan cara diatas hasil perkalian 2 dan modulo dengan kunci polinomial G(x) pada persamaan (2) yang digunakan pada CRC-32 dapat dilakukan dalam 1 siklus. Sehingga didapatkan pola output untuk masing-masing digit sesuai Gambar 2. Dengan menginputkan hasil operasi ini berulang kali sampai diperoleh sisa hasil bagi yang sesuai akan didapatkankode CRC yang akan digunakan untuk skema pengecekan kesalahan transmisi data.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (2)$$

*2.2. Implementasi Sistem*

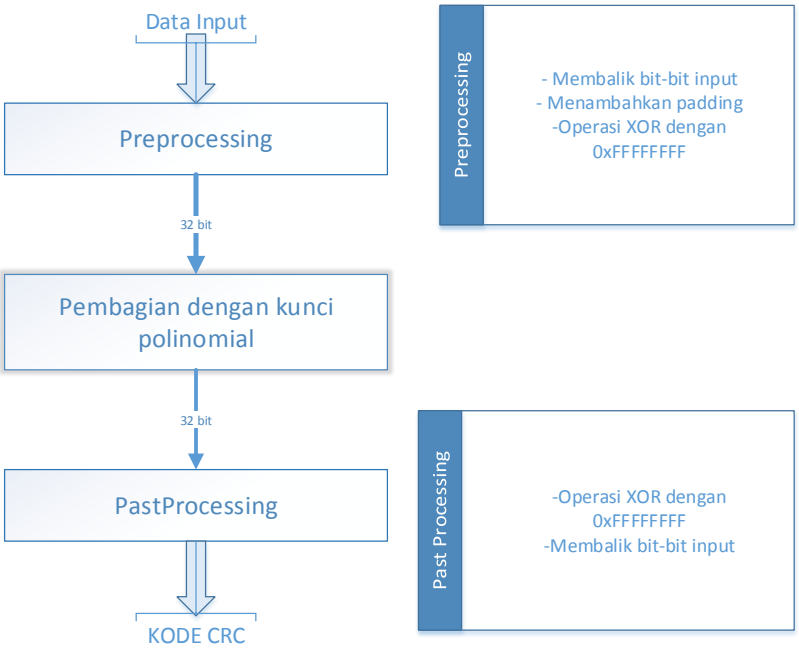
Gambar 4 memperlihatkan kode VHDL dari unit dasar pembagian yang digunakan untuk merealisasikan komputasi perkalian 2 sebuah masukan dan sekaligus menghitung modulasi terhadap kunci polinomial G(x) pada persamaan (2). Unit ini yang nantinya digunakan untuk mendapatkan hasil dari tahapan pembagian dari komputasi kode CRC.

Tahapan terakhir dari komputasi kode CRC adalah tahapan pastprocessing. Sebenarnya tahapan ini adalah tahapan penyelesaian yang dibagi menjadi 2 langkah. Langkah pertama adalah melakukan operasi XOR terhadap output yang didapatkan pada tahap pembagian bilangan dengan kunci polinomial. Pada langkah selanjutnya keluaran dari hasil operasi XOR dibalik nilainya, yang tadinya berada pada posisi MSB sekarang berada pada posisi LSB. Hal ini terjadi untuk keseluruhan 32 bit sinyal yang diperoleh.

```
Y(31 downto 27) <= X(30 downto 26);
y(26) <= x(31) XOR x(25);
y(25 downto 24) <= X(24 downto 23);
y(23) <= x(31) XOR x(22);
y(22) <= x(31) XOR x(21);
y(21 downto 17) <= x(20 downto 16);
y(16) <= x(31) XOR x(15);
y(15 downto 13) <= x(14 downto 12);
y(12) <= x(31) XOR x(11);
y(11) <= x(31) XOR x(10);
y(10) <= x(31) XOR x(9);
y(9) <= X(8);
y(8) <= x(31) XOR x(7);
y(7) <= x(31) XOR x(6);
y(6) <= x(5);
y(5) <= x(31) XOR x(4);
y(4) <= x(31) XOR x(3);
y(3) <= x(2);
y(2) <= x(31) XOR x(1);
y(1) <= x(31) XOR x(0);
y(0) <= x(31);
```

Gambar 4 Kode untuk unit dasar pembagian

Dari penjelasan diatas dapat diketahui bahwa sangatlah mungkin untuk melakukan komputasi kode CRC dalam satu siklus. Karena semua hasil operasi merupakan operasi kombinasional. Aliran data yang tejadi pada operasi komputasi kode CRC sendiri diperlihatkan Gambar 5.




Gambar 5 Rancangan aliran data generator CRC

## 3. HASIL DAN PEMBAHASAN

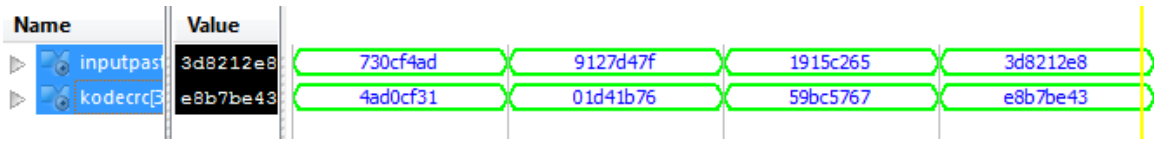
Pada penelitian ini dilakukan verifikasi fungsional perangkat kerang menggunakan bantuan simulatoryang disediakan tools perancangan Xilinx ISE Design Suite 14.5. Setelah hasil verifikasi fungsional dinyatakan benar, selanjutnya akan dilakukan inplementasi fisik pada FPGA keluarga Xilinx Spartan®-6 Seri (XC6LX16-CS324).

### *3.1. Verifikasi Fungsional Sistem*

Verifikasi fungsional sistem diawali dengan menguji ketiga unit pembagun rangkaian CRC generator sesuai dengan 3 tahapan utama yang dibutuhkan untuk melakukan komputasi kode CRC. Gambar 6 menunjukkan hasil simulasi unit preprocessing. Gambar 7 merupakan hasil simulasi untuk unit pembagian yang merupakan unit inti pada komputasi kode CRC. Hasil verifikasi fungsional untuk unit past processing ditampilkan pada Gambar 8. Dari ketiga gambar hasil verifikasi fungsional dapat dilihat bahwa setiap unit mampu menyelesaikan komputasinya masing-masing dalam 1 siklus detak. Hal ini dapat dilihat dengan langsungnya terjadi perubahan output pada siklus yang sama saat terjadinya perubahan input.

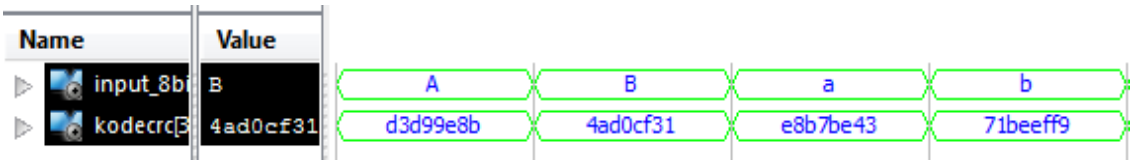

Gambar 6 Simulasi unit prerocessing


Gambar 7 Simulasi unit pembagian




Gambar 8 Simulasi unit past processing

Gambar 9 memperlihatkan hasil simulasi fungsional keseluruhan sistem dengan mengintegrasikan ke 3 unit modul yang terdapat dalam generator kode CRC. Setelah integrasi dilakukan, hasil pengujian menggunakan simulator ISIM yang terdapat pada toolsperancangan Xilinx ISE Design Suite 14.5 menunjukkan bahwa keseluruan komputasi dapat dilakukan dalam 1 siklus detak. Hal ini ditegaskan pada Gambar 8 bahwa kode CRC yang didapat untuk kode ASCII karakter "A" adalah "d3d99e8b".




Gambar 9 simulasi keseluruhan

*3.2. Pengukuran kinerja perangkat keras*

Setelah perangkat keras yang dihasilkan diuji secara fungsional, maka tahap selanjutnya adalah mengukur kinerja perangkat keras yang dihasilkan tersebut, yaitu; kecepatan dan pemakaian sumber daya. Pengukuran kecepatan ditentukan menggunakan 3 paramater yang harus diperhatikan, yaitu:

- Latensi.
  Latensi merupakan interval waktu antara data dimasukkan dengan data selesai diproses, yang biasanya dihitung dalam satuan siklus detak. Berdasarkan hasil verifikasi fisik yang dilakukan perangkat ini mampu menghasikan kode CRC dalam 1 siklus detak, sesuai dengan proses perancangan.
- Pewaktuan.
  Pewaktuan (timing) didefenisikan sebagai interval waktu antara siklus detak.Karena unit perangkat keras bekerja pada rangkaian kombinasional, maka untuk menentukan frekuensi kerja maksimum perangkat ini dapat dilihat dari jalur kritis yang diproleh dari hasil implementasi ke FPGA. Jalur kritis terpanjang pada rangkaian ini berdasarkan laporan hasil implementasi yang dilakukan oleh Xilinx ISE Design Suite 14.5 yaitu sebanyak 8,515ns dengan demikian dapat disimpulkan bahwa periode kerja perangkat keras tidak akan melebihi 8,515ns tersebut. Sehingga frekuensi kerja maksimum adalah sekitar 117,44 MHz pada FPGA keluarga Xilinx Spartan®-6 Seri (XC6LX16-CS324) .
- Lewatan (throughput)
  Jumlah dalam 1 satuan waktu (biasanya dalam 1 detik) dikenal dengan nama lewatan. Lewatan dapat dihitung berdasarkan latensi dan pewaktuan yang ada menggunakan rumus (3). Jumlah 1 blok data yang dihasilkan pada 1 kali perhitungan adalah sebanyak 32 bit dan latensi yang dibutuhkan adalah 1 saja, maka hasil perhitungan yang didapatkan untuk lewatan ini adalah sebesar 3,758 Gbps.

$$lewatan = \frac{Frekuensi\ kerja\ Maksimal\ x\ jumlah\ 1\ blok\ data}{Latensi} \quad (3)$$

Pengukuran pemakaian sumber daya dilakukan dengan menghitung jumlah slice (CLB) dan Pin I/O yang digunakan pada implementasi ini. Jumlah Slice yang digunakan pada realisasi FPGA ini adalah 10 slice dengan total 16 LUT yang tersedia yang digunakan. Sedangkan jumlah pin masukan yang digunakan 8 pin . Pin Keluaran yang digunakan sebanyak 32, sehingga total penggunaan pin I/O adalah 40 pin. Semakin banyak sumber daya FPGA digunakan untuk menyelesaikan komputasi maka semakin banyak daya yang dikonsumsi. Jika dilihat dari sisi persentase resource yang terpakai masih sedikit sekali, sehingga masih sangat mungkin untuk mengimplementasikan lebih dari 50 perangkat keras yang sama pada FPGA tipe ini dengan menduplikasi rangkaian generator kode CRC ini karena rangkaian ini hanya menggunakan 1% dari keseluruhan perangkat keras yang tersedia pada FPGA Xilinx Spartan 6 seperti yang ditunjukkan Tabel 1. Dari tabel dapat dilihat bahwa semua komponen yang digunakan adalah komponen LUT yang terdapat pada Slice, sedangkan Flip-flop yang terdapat pada slice tidak digunakan untuk 10 slice yang dimanfaatkan untuk rangkaian generator CRC ini. Berdasarkan hasil ini dapat dibuktikan bahwa rangkaian Generator CRC dapat sepenuhnya direalisasikan menggunakan rangkaian kombinasional.

Tabel 1 Pemakaian sumber daya FPGA

| Komponen | Pemakaian | Utilisasi |
|---|---|---|
| Slice | 10 | 1% |
| Slice as Logic | 16 | 1% |
| LUT 6 | 4 | - |
| LUT 5 dan 6 | 12 | - |
| Unused FF | 16 | 16% |
| Bonded I/O | 40 | 17% |

Seberapa optimal implementasi yang dilakukan dapat diketahui dengan menghitung efisiensi (throughput per slice). Lewatan (throughput) dari perangkat keras ini adalah 3,758 Gbps dan jumlah slice yang digunakan untuk merealisasikan arsitektur ini pada FPGA Xilinx Spartan 6 adalah 10 slice maka efisiensi dari rancangan ini adalah 0,38Gbps/Slice. Tabel 2 menunjukkan kinerja rangkaian CRC yang diimplementasikan pada beberapa tipe FPGA. Hasil implementasi menunjukkan bahwa penggunaan slice yang paling sedikit adalah pada FPGA Spartan®-6 dibandingkan beberapa tipe FPGA lain. Jika dibandingkan dari sisi arsitektur slice dari ke 3 FPGA, maka Slice pada FPGA tipe Spartan 6 memiliki jumlah input LUT yang paling besar. Hal ini sangat sesuai dengan arsitektur rangkaian generator CRC yang diimplementasikan karena memang membutuhkan LUT dengan input yang besar. Dari tabel 2 juga dapat dilihat FPGA tipe Virtex 5 memiliki lewatan yang paling besar dibandingkan hasil implementasi pada 2 tipe FPGA lain. Hasil ini didapatkan karena implementasi rangkaian CRC Generator pada FPGA ini memiliki pewaktuan setengah pewaktuan dari hasil implementasi rangkaian CRC lain. Oleh karena itu, efisiensi yang diperoleh untuk implementasi rangkaian CRC Generator pada tipe ini adalah efisiensi yang paling besar.

Tabel 2 Kinerja perangkat keras pada beberapa tipe FPGA

| **Jenis FPGA** | **Spartan®-6** | **Spartan®-3** | **Virtex®-5** |
|---|---|---|---|
| **Timing (ns)** | 8,515ns | 8,122 | 4,385 |
| **Frek. Mak. (MHz)** | 117,44 | 123,12 | 228,050 |
| **Latensi (Siklus)** | 1 | 1 | 1 |
| **Lewatan (Gbps)** | 3,758 | 3,94 | 7,30 |
| **Slice** | 10 | 16 | 16 |
| **Efisiensi(Gbps/Slice)** | 0,38 | 0,25 | 0.46 |

Tabel 3 memperlihatkan perbandingan hasil implementasi rangkaian CRC Generator pada FPGA spartan 3 di penelitian ini dengan penelitian [7]. Tabel 3 menunjukkan bahwa hasil

implementasi rangkaian CRC Generator menggunakan metode pada penelitian ini, memanfaatkan sumber daya yang jauh lebih sedikit dibandingkan metode LFSR yang selama merupakan metode yang menghasilkan sumber daya paling minimal. Hasil ini dapat dikarenakan metode LFSR sendiri membutuhkan banyak register untuk merealisasikan shift register untuk melakukan operasi pembagian agar diperoleh kode CRC. Sedangkan pada setiap Slice FPGA tipe Spartan 3, hanya terdapat 2 register Flip-flop dan 2 SLR saja.

Tabel 3 Perbandingan kinerja rangkaian dengan penelitian lain

| Komponen perbandingan | Peneliti [7] | Penelitian ini |
|---|---|---|
| Metode | LFSR | Pemisahan digit koefisien |
| Timing | 8,097 ns | 8,122 ns |
| Frekuensi Maksimal | 117,08 MHz | 123,12 MHz |
| Latensi | 32 | 1 |
| Jumlah Slice | 56 | 16 |
| Jumlah Slice FF | 71 | 0 |
| Bonded I/O | 26 | 40 |

Jika dilihat dari sisi kecepatan, metode LFSR memang memiliki kecepatan yang lebih lambat dibandingkan metode lain karena latensi pada metode ini akan cendrung meningkat seiring dengan pertambahan derajat polinomikan kunci yang digunakan untuk CRC. Walaupun frekuensi kerja maksimal yang didapatkan dari hasil implementasi metode ini tetap besar seperti yang diperlihatkan pada Tabel 3, namun karena latensinya yang banyak, menyebabkan metode LFSR memiliki lewatan yang sangat kecil dibandingkan metode yang diiusulkan pada penelitian ini.

## 4. KESIMPULAN

Hasil pengujian menunjukkan telah berhasil diimplementasikan Rangkaian CRC Generator pada FPGA yang mampu menyelesaikan komputasi dalam 1 siklus detak. Lama pewaktuan yang diperoleh untuk implementasi ini adalah 8,515 ns dengan jumlah lewatan yang dicapai sebesar 3,758 Gbps. Hasil implementasi ditanam pada FPGA Xilinx Spartan 6 pada board Nexys 3 memanfaat sumber daya kurang dari 1% dari sumber daya yang tersedia dengan rincian 10 Slice dan 40 Pin I/O. Dengan demikian efisiensi penggunaan resource yang didapat adalah 0, 38Gbps/Slice.

## 5. SARAN

Penelitian ini adalah tahap awal dari implementasi CRC generator pada FPGA dimana belum dilakukan proses duplikasi rangkaian perangkat keras yang dihasilkan untuk memfasilitasi perhitungan untuk aliran data kecepatan yang lebih tinggi dari. Ke depan akan dikembangkan CRC-Generator yang mampu menghasikan kode CRC untuk beberapa jenis data dan dapat diimplementasikan pada data stream untuk menguji kinerja dari perangkat keras secara keseluruhan. Selain itu metode mengurangi latensi perhitungan dengan memisahkan koefisien digit-digit tertentu dan menghitung langsung hasil modulo dengan polinomial kunci yang digunakan pada penelitian ini dapat juga diimplementasikan untuk beberapa tipe CRC yang ada walaupun menggunakan polinomial yang berbeda.

## UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Departemen Ilmu Komputer dan Elektronika yang telah memberi dukungan terhadap penelitian ini.

DAFTAR PUSTAKA

[1] Mitra, J. dan Nayak, T., 2017, Reconfigurable very high throughput low latency VLSI (FPGA) design architecture of CRC 32, *The VLSI Journal*, 56, pp. 1-14.[Online]. Available:https://www.sciencedirect.com/science/article/pii/S0167926016300669. [Accessed: 18-Maret-2018]

[2] S. Panda and G. L. Kumar, "Comparison of serial data-input CRC and parallel data-input CRC design for CRC-8 ATM HEC employing MLFSR," *2014 International Conference on Electronics and Communication Systems* (ICECS), Coimbatore, 2014, pp. 1-4. [Online].Available:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6892739&isnumber=6892507. [Accessed: 10-Feb-2019]

[3] C. E. Kennedy and M. Mozaffari-Kermani, "Generalized parallel CRC computation on FPGA," *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, Halifax, NS, 2015, pp. 107-113.[Online] Available : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7129169 [Accessed : 20 Oktober 2018]

[4] A. R. Buzdar, L. Sun, R. Kashif, M. W. Azhar and M. I. Khan, Cyclic Redundancy Checking (CRC) Accelerator for Embedded Processor Datapaths, "*International J. of Advanced Computer Science and Applications*, Vol 8, No. 2, pp 321- 325, 2017[Online]. Available:https://thesai.org/Downloads/Volume8No2/Paper_42-Cyclic_Redundancy_Checking_CRC_Accelerator.pdf. [Accessed: 14-Mare-2018]

[5] Y. Jun, D. Jun, L. Na, G. Yixiong and D. Yin, "FPGA-based multi-channel CRC generator implementation," *2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT)*, Shenzhen, 2010, pp. 81-84. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5496514&isnumber=5496503. [Accessed: 08-April-2017]

[6] G. Sowndharya and A. Vasuki, "Reducing bit error rate using CRC verification in turbo codes," *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, 2017, pp. 627-631. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8299834. [Accessed: 19-Oktober-2018]

[7] M. F. Hasmi, dan A. G. Keskar, "An Optimized FPGA Implementation of CAN 2.0 Protocol Error Detection Circuitry, *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 6, No. 3, pp. 602-614. Jun. 2017 [Online]. Available: https://www.iaescore.com/journals/index.php/IJEECS/article/download/7398/6616. [Accessed: 02-April-2018]

[8] S.N.V.P.Kumar, S. B. Jyothi, G. K. S. Tejaswi, FPGA Based Design Of Parallel CRCGeneration For High Speed Application, *IJSRET (International Journal of Scientific Research Engineering & Technology*, Vol 6, 3, pp 258- 264, Mar. 2017 [Online]. Available: http://www.ijsret.org/pdf/121757.pdf. [Accessed: 20-Maret-2018]

[9] Y. Huo, X. Li, W. Wang and D. Liu, "High performance table-based architecture for parallel CRC calculation," The 21st IEEE International Workshop on Local and Metropolitan Area Networks, Beijing, 2015, pp. 1-6. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7114717&isnumber=7114713. [Accessed: 08-April-2018]

[10] S. Ghaznavi, C. Gebotys and R. Elbaz, "Efficient Technique for the FPGA Implementation of the AES MixColumns Transformation," 2009 International Conference on Reconfigurable Computing and FPGAs, Quintana Roo, 2009, pp. 219-224.[Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5382055&isnumber=5381991. [Accessed: 02-April-2018**]**